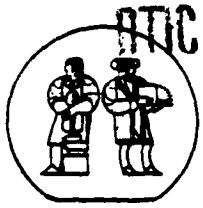

(4)

Massachusetts Institute of Technology | Microsystems Research Center | Cambridge Massachusetts 02139 | Room 39-321 Telephone (617) 253-8138

VLSI Memo No. 88-449
June 1988

THE RECONFIGURABLE ARITHMETIC PROCESSOR

Stuart Fiske and William J. Dally

Abstract

The Reconfigurable Arithmetic Processor (RAP) is an arithmetic processing node for a message-passing, MIMD concurrent computer. It incorporates on one chip several serial, 64 bit floating point arithmetic units connected by a switching network. By sequencing the switch through different patterns, the RAP chip calculates complete arithmetic formulas. By chaining together its arithmetic units the RAP reduces the amount of off chip data transfer; In the examples we have simulated off chip I/O can often be reduced to 30% or 40% of that required by a conventional arithmetic chip. Simulations predict a peak performance of 20M Flops with 800M bit/sec off chip bandwidth in a 2 μm CMOS process.

Acknowledgements

This research was supported in part by the Defense Advanced Research Projects Agency under contract nos. N00014-80-C-0622 and N00014-85-K-0214, an NSF Presidential Young Investigators Award, and by a scholarship from the Fonds pour la Formation de Chercheurs et l'Aide a la Recherche (Fonds FCAR).

Author Information

Fiske and Dally: Department of Electrical Engineering and Computer Science, Artificial Intelligence Lab, MIT, Cambridge, MA 02139; Fiske: Room NE43-416, (617) 253-8473; Dally: Room NE43-417, (617) 253-6043.

The Reconfigurable Arithmetic Processor [1]

Stuart Fiske and William J. Dally
Artificial Intelligence Laboratory and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

The Reconfigurable Arithmetic Processor (RAP) is an arithmetic processing node for a message-passing, MIMD concurrent computer. It incorporates on one chip several serial, 64 bit floating point arithmetic units connected by a switching network. By sequencing the switch through different patterns, the RAP chip calculates complete arithmetic formulas. By chaining together its arithmetic units the RAP reduces the amount of off chip data transfer: in the examples we have simulated off chip I/O can often be reduced to 30% or 40% of that required by a conventional arithmetic chip. Simulations predict a peak performance of 20MFlops with 800Mbit/sec off chip bandwidth in a 2μm CMOS process.

1 Introduction

1.1 Summary

The problem in building fast arithmetic chips does not have to do as much with building fast arithmetic circuits as with being able to supply the necessary I/O bandwidth. For example, a conventional 64 bit-parallel floating point adder or multiplier pipe running at 20MFlops requires an I/O bandwidth of 3.8Gbit/sec to be kept busy. This level of I/O is very difficult to achieve with anything less than dedicated lines and a continuous stream of data.

The Reconfigurable Arithmetic Processor (RAP) is a CMOS, 64 bit, floating point arithmetic chip designed to sustain high rates of floating point operations, while requiring only a fraction of the I/O bandwidth of a conventional arithmetic chip. To do this the RAP allows the direct calculation of expressions that contain several adds, subtracts, and multiplies. In effect the chip can be thought of as calculating a complete formula rather than a single primitive operation.

The RAP uses serial arithmetic. Serial implementations of arithmetic are more area efficient than parallel implementations and allow us to put several Arithmetic Units (AUs) on a single chip. Having narrow serial datapaths also allows us to implement an area efficient switching network that can be used to route data between AUs. Although a single serial unit is slower than a parallel implementation, the RAP makes up for this by exploiting the functional parallelism achieved by having several units on one chip: instead of a single 20MFlop unit, we have eight 2.5MFlop units running in parallel. In the examples we have simulated, AU utilization ranged from 30% to 60% depending on the problem.

The RAP datapath shown in Figure 1 consists of a number of four-bit serial AUs, a switch, input registers, and output registers. Data first enters the switch and gets routed to the appropriate AUs. Intermediate results are fed back into the switch which is reconfigured to allow the next stage of the computation to take place. When the computation is complete the results are sent to the output registers.

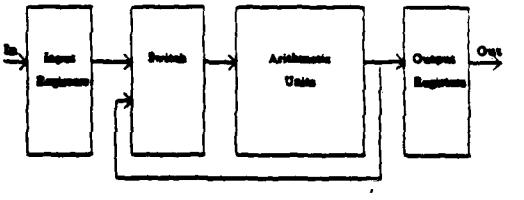

Figure 1: RAP Datapath

At a higher level, the RAP has a message passing interface. A RAP is sent messages that define equations as a sequence of switch configurations, which are stored in local memory. Subsequent messages use these stored configurations to evaluate the equation. Mechanisms are included that allow the pipelining of several RAPs so that the output of one RAP can be used as the input to another.

1.2 Background

Numerically intensive computer applications such as analog circuit simulation, the simulation of physical phenomena such as N-body problems and finite element analysis, digital signal processing, and three dimensional graphics require large amounts of floating point computing power [15]. To satisfy this demand, many special purpose board level and chip level arithmetic processors have been built [7] [11] [17]. Approaches range from math coprocessors that act as simple extensions to a main processor (e.g. the Intel 80387 and the Motorola MC68881) to dedicated math processors designed for specific applications. In most cases these processors are implemented using a bit-parallel approach. Because of this approach, implementations are expensive in terms

[1] The research described in this paper was supported in part by the Defense Advanced Research Projects Agency under contract N00014-88-C-0622 and N00014-88-K0124, in part by an NSF Presidential Young Investigator Award, and in part by a scholarship from the Fonds pour la Formation de Chercheurs et l'Aide a la Recherche (Fonds FCAR).

of silicon area and only one or two floating point units can be put on a single chip.

The area efficiency of serial arithmetic allows several floating point units to be put on a single chip. Serial arithmetic has been used in many Digital Signal Processing applications [6] [8]. The idea of exploiting functional parallelism using serial fixed point arithmetic has been used in this area [9]. Many algorithm alternatives exist for serial arithmetic implementation [1] [10] [13] [16].

Another approach to the I/O bandwidth problem is to use a register file to store operands and intermediate results [17]. The register file serves the same function as a switch, selecting data to be input to each function unit during each pipeline time slot. The register file performs this switching both by storing data to move it to a different time slot, and by multiplexing many registers into each register file port. The serial switch in the RAP eliminates the need for storage and simplifies the multiplexing. The resulting switch is smaller, both because it is serial and because it contains no storage. The switch is also simpler to control: switch configurations are changed each word time, while register file addresses must be changed each clock cycle. The slow control signals allow us to operate the switch faster than a comparable register file. Throughout the rest of this paper, the bit-parallel AU with no means of exploiting locality is used as a baseline level of comparison.

The RAP chip is being designed as a part of the J-Machine [3], a message passing concurrent computer system under development at MIT. This system is based on a mesh routing network that connects a collection of processing nodes, and uses wormhole routing techniques to reduce message latency to 2μs for a 200 bit message on a 4K node network [5]. Each single-chip node includes both the network communication hardware and a processor. The RAP chip is one node type that can fit into the network "slots". It includes the necessary control mechanisms and message handling capabilities to fit into the system. The RAP borrows several ideas that were first developed in the Message Driven Processor (MDP) [2] which is the general purpose computing node for the system. In particular the RAP executes messages directly, reducing message interpretation overhead, and it makes use of the same network communication scheme [4] [5].

1.3 Outline

The remainder of this paper describes the RAP in detail. Section 2 gives an example of how the RAP is used in a typical application. Section 3 describes the messages that are used to control the RAP, while section 4 describes the architecture. Performance results derived from simulations are presented in section 5. Section 6 briefly discusses a simplified fixed-point RAP chip that has been fabricated and tested. Finally, section 7 offers some concluding remarks.

2 An example

In order to illustrate how the RAP uses functional parallelism to exploit the locality and concurrency found in mathematical equations, we consider the calculation a 4-point Fast Fourier Transform (FFT) [12] [14]. The 4-point FFT dataflow graph is shown in Figure 2 and consists in 12 multiplies and 22 additions used to calculate the real and imaginary parts of the 4 output results. This graph would be evaluated by a RAP as follows: First a "method" would be stored in the RAP memory describing each level of the calculation. Then a message would be sent containing the 14 input variables necessary for the computation. Assuming an ideal setting, the RAP would successively run through each level of the calculation as described by the method, exploiting functional parallelism by doing all operations of a given level in parallel. Finally it would send a message containing the results to the appropriate destination.

In a realistic setting, determining the successive configurations of a method involves a scheduling problem, since the RAP may not have enough AUs to perform all possible concurrent operations at once. The RAP we are building has 4 adders/subtractors and 4 multipliers. An optimum schedule for the graph of Figure 2 is shown in Figure 3: the 4-point FFT actually requires 7 computation cycles. In this application a 60% of the RAPs capacity is used for a rate of 12 MFlops.

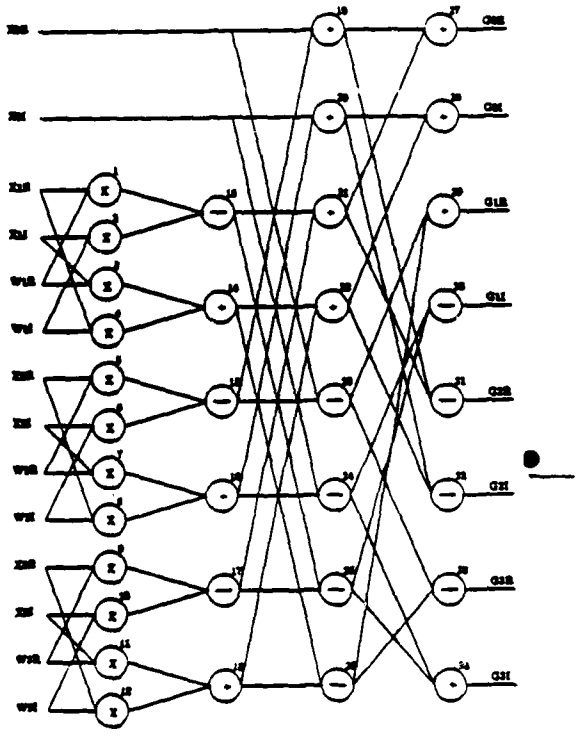

Figure 2: 4-Point FFT Dataflow Graph

The I/O bandwidth required is reduced to 25% of the bandwidth required by a conventional bit-parallel arithmetic chip. A conventional arithmetic chip would require $34 \times 3 = 102$ word transfers,

where 34 corresponds to the number of operations, and 3 corresponds to the two words of input data and one word of output data for each operation. Using a RAP, only 26 words must be transferred on and off chip, consisting of 14 input operands, 8 output results, and 4 words of overhead information. This reduced I/O bandwidth makes it possible for a communications network to keep the chip busy.

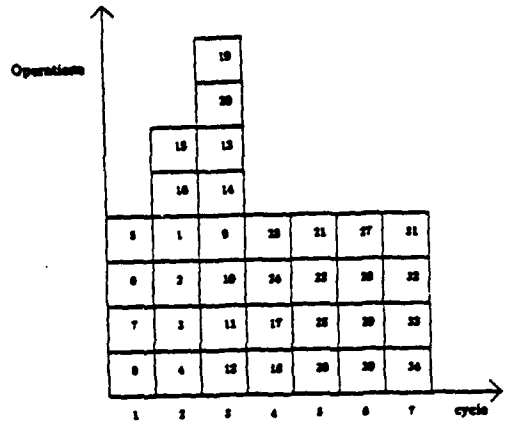

Figure 3: Operation Schedule for the operations of the 4-point FFT Dataflow Graph

3 Messages

There are three types of messages that the RAP processes in order to support the types of operations described above:

1. CONFIGURE AND EXECUTE (C+E). This message causes operands to be loaded into the input registers, passed through one or more switch configurations, and then unloaded from the output registers. This is repeated for each set of operands in the message.
2. STORE METHOD (SM). This message is used to store a method in local memory so that it can be used by the C+E message. A method describes a sequence of switch configurations necessary to perform a calculation.
3. STORE TEMPLATE (ST). This message is used to store a template in local memory. A template contains forwarding information that allows the cascading of several RAP chips.

Message formats are shown in Figure 4. The C+E message has METHOD-ID and TEMPLATE-ID fields that specify the method and template to be used. Method and template IDs are memory addresses that point at the first element of the method or template. Methods and templates must be sent to the RAP before the C+E commands that use them. The C+E message also has NODE-ID and REPLY-ID fields that specify the ultimate destination of the results. The NODE-ID is the network address of a non-RAP node, and the REPLY-ID is a message header. These two fields are used in conjunction with template information to forward output results. The information contained in methods and templates is discussed in detail in the following sections.

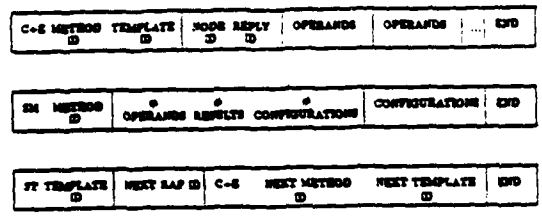

Figure 4: Message Formats

3.1 Methods

A method consists of all the information necessary to put operands through a sequence of switch configurations. It includes:

1. Which input registers to load with the operands.
2. Which output registers will contain the results.
3. The number of switch configurations that the operands are to go through.
4. A description of each of the configurations. A configuration contains bits describing the switch connectivity, and bits that determine the functionality of the AUs (e.g. a bit might determine whether an AU does an add or a subtract).

The first three pieces of information are packed into one word (64 bits) of the method description. Each configuration also takes one word. The number of sets of operands that will be passed through using a given method is not included in the method description since it can deduced from the end of message signal.

3.2 Templates

A template is used to permit the cascading of several RAP chips. It contains information that allows the forwarding of the output data, in the form of a C+E message, to another RAP for further computation. A template consists of the address of the next RAP that the results are to be sent to, and the instruction (method and template) that is to be executed there.

Cascading of RAPs works as follows: A general purpose node such as a MDP [2] sets up the pipeline by loading methods and templates into the appropriate RAP chips. A C+E message is then sent to the first RAP in the pipeline to begin the calculation. Each RAP in the pipeline uses its template to forward its results to the next RAP for the next stage of the computation. The NODE-ID and REPLY-ID are passed from RAP to RAP until they are used at the last stage to get the results to their final

destination. In some cases it is necessary to combine results coming from different RAPs before continuing the computation: this combining can be done by an MDP.

Templates are specified separately from the method to allow different calculations to use common subroutines and to permit a single calculation to distribute its work over several RAPs. An example of two calculations using a common subroutine would be a routine that multiplies all the elements of two vectors. Depending on which template is used to forward the result, such a routine could be used by itself or could be used in a dot product routine where the next step is to add all the products together. A RAP could also use the different templates to divide up a problem so that the next stage of the calculation is being executed on a number of different RAPs. In this case several RAPs would contain the same method and the choice of templates would distribute the work over these processors.

4 Architecture

4.1 Block Diagram

Figure 5 shows a block diagram of the complete RAP consisting of the control blocks, the memories, and the datapath. There are three control blocks referred to as input control, output control, and switch control, that coordinate the execution of messages. Input control handles the reception of messages, the input to the datapath, and memory operations. Output control is responsible for unloading result messages into the output queue, while switch control is responsible for loading switch configurations at the correct time. By dividing the control into these three different blocks the operations of loading operands, unloading results, and changing the switch configuration can be pipelined. Hardware interlocks resolve memory contention and provide feedback to prevent the output queue from overflowing.

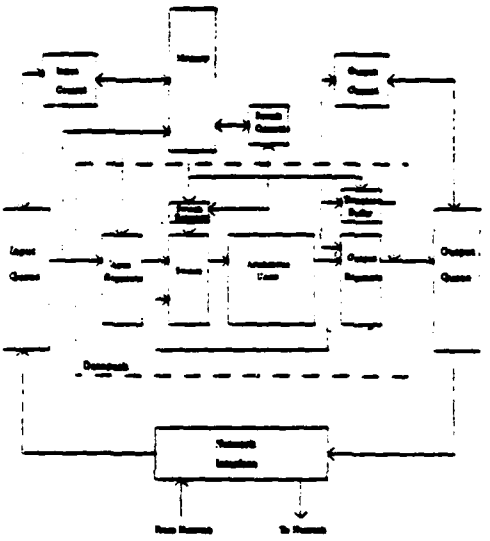

Figure 5: RAP Block Diagram

There are three memories on the chip: a main memory for holding templates and methods, an input queue, and an output queue The input and output queues are 64 word memories with separa ports for the network and processor. The main memory (256 words) is shared between the input control and the switch control, with priority given to the switch control.

The datapath consists of 16 input registers, a switch, a switch configuration register, a collection of 16 function units, 16 output registers, and some buffer storage for the template. The 16 function units consist of 4 plus/minus arithmetic units (AUs), 4 multiply AUs, and 8 feedthroughs used to pass operands unchanged with the appropriate delay.

Operands are loaded into the input registers from the input queue, propagate through the function units for one or more "computation cycles" (which is defined to be the time for one pass through the switch and functional units) with the outputs of the AUs and feedthroughs feeding back into the switch, and then are unloaded from the output registers into the output queue. The input and output registers perform parallel-serial and serial-parallel conversion respectively, and can be loaded or unloaded as the switch and AUs are busy computing another problem instance. After each computation cycle the switch is reconfigured by the switch control unit which reloads the switch configuration register. The appropriate template is unloaded into the output queue before the output results.

4.2 Arithmetic Units

The RAP includes adder/subtractors, and multipliers. We estimate that these units will run at 80 Mhz in a 2μm CMOS process. For simplicity reasons, a non-standard floating point format was chosen, consisting of an 8 bit, two's complement exponent field, and a 56 bit, two's complement mantissa field. This format permits a uniform treatment of the exponent and mantissa in two's complement form. The implementation is four-bit serial in order to make full use of the clock period. In single bit implementations signals propagate in times much smaller than the smallest clock period that can be reliably distributed, and thus do not make full use of the clock period. Manipulating two or four bits at a time also allows certain efficient serial algorithms to be used [1] [10]. Area is approximated to be $4M\lambda^2$ for an AU and $500K\lambda^2$ for a feedthrough.

The AUs run four times faster than the memory. We denote a memory cycle as a major cycle and an AU cycle as a minor cycle. A word time is defined as the time required to shift a complete operand into an AU, and corresponds to 16 minor cycles or 4 major cycles. The units have a latency of two word times in which the exponent and mantissa are computed, and normalization is performed.

4.3 Switch

The switch is shown in Figure 6. Each AU selects one of 8 inputs for each of their two operands, while the feedthroughs each have the choice of 2 inputs. On the first configuration of any given method the inputs are taken from the 16 input registers, while on subsequent configurations the inputs are taken from the outputs of the AUs and feedthroughs. The column of 2X1 multiplexers is used to make this choice.

The switch chosen does not offer the complete connectivity of a 16X16 crossbar but has the advantage of being much smaller and requiring less state information: each configuration can be described by using 60 bits (3 bits for each AU input, 1 bit for each feedthrough, and 1 bit for each adder/subtractor unit to select its function) which fits into a single 64 bit word. This allows a change of the switch configuration in a single clock cycle by reading a single word from memory. For the problems we simulated, the incomplete connectivity did not prevent us from mapping the problems efficiently onto the switch. The main reason for this was that during any given stage of the calculation not all the AUs are needed, so that it is easy to route outputs to the desired inputs by choosing between several possible free AUs.

An expression compiler is needed to map a given equation or set of equations into a series of appropriate switch configurations. A compiler is currently under development based on a critical path analysis of the expression and a greedy scheduling of operations.

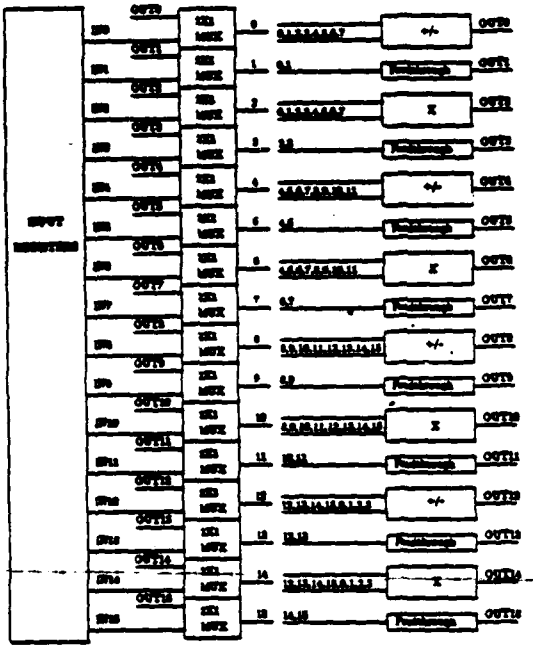

Figure 6: Switch Configuration

5 Performance Evaluation

A simulator for the RAP architecture has been written and used to verify control and to evaluate performance. This simulator allows the sending of messages to the RAP, as well as the cascading of RAPs. Performance figures assume a minor cycle of 12.5ns, a major cycle of 50ns, an input bandwidth of 400Mbit/sec, and an output bandwidth of 400Mbit/sec [5].

A number of formulas have been mapped into the RAP and some of these results are shown in Table 1. For each problem in this table we list the total number of floating point operations performed, the number of input operands and output operands, and the number of switch configurations in the method used to define the calculation. From these figures we calculate the average floating point rate achieved and the I/O bandwidth required to keep a RAP busy with the problem. The latency column refers to the time from when one problem instance is in the input buffer to when the complete result is in the output buffer and includes all control overhead.

Average floating point performance achieved depends on how well a problem is able to use the parallelism made available by the RAP. The average floating point performance for these problems was 9MFlops or 45% of the peak performance possible. Optimization is possible in some of these cases by combining several problems in order to use more of the resources: for instance doing two 2X2 FFTs at the same time uses the RAP more efficiently than a single 2X2FFT. The I/O bandwidth required depends on the locality inherent in the problem. For instance the vector sum example has no locality that can be exploited by the RAP and there is no bandwidth advantage in using it (in fact if overhead is included, using the RAP is more costly). The bandwidth required for the other problems however have all been reduced to within the capacity of our network. The I/O bandwidth required will increase slightly because of communication and message handling overhead. The percentage cost of this overhead depends on how many sets of operands are sent in a single message.

Problem	# Ops.	I/O Words (input + output)	# Configs.	Floating Point Rate	Latency	I/O Bandwidth Required
Vector Sum	8	16 + 8	2	18MFlops	2.83μs	169Mbit/s
Vector Dot Product	15	16 + 1	5	7.5MFlops	3.75μs	544Mbit/s
Van der Pol's Equation	20	8 + 4	8	8.25MFlops	4.84μs	240Mbit/s
2X2 FFT	16	8 + 4	3	8.2MFlops	2.5μs	533Mbit/s
Two 2X2 FFTs	28	12 + 8	5	16MFlops	3.83μs	640Mbit/s
4X4 FFT	34	14 + 8	7	12MFlops	4.75μs	505Mbit/s

Table 1: RAP performance in typical applications

The key feature of the RAP is that it reduces the data transfer bandwidth that the network must sustain to do arithmetic calculations efficiently. Matching the network speed to the RAP speed is an important consideration. To achieve balance, each method should have sufficient configurations to keep the RAP busy without overloading the network, and few enough for the

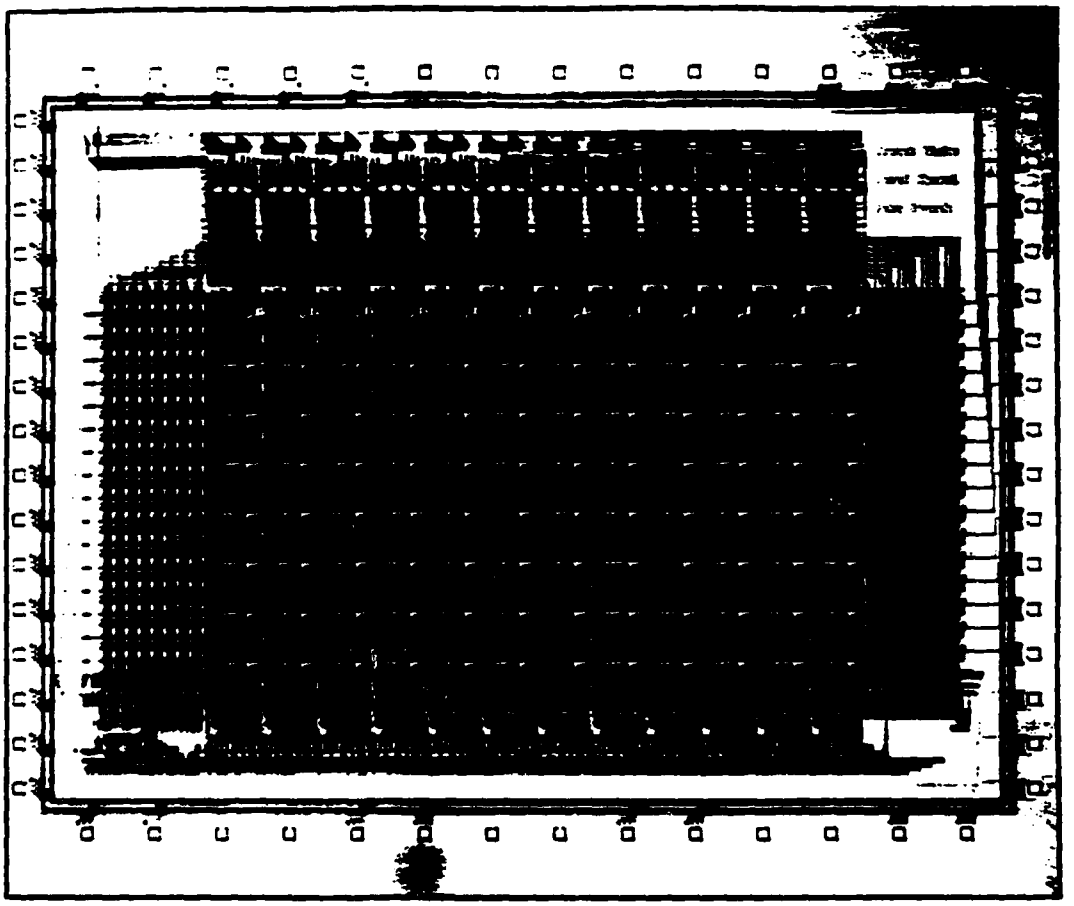

Figure 7: RAP Prototype

RAP to keep up with the network. Table 2 shows how many configurations in a method would be ideal, given the number of input operands. Note these figures assume an unloaded network: if there is other traffic on the network the I/O bandwidth will decrease, the time for a complete set of operands to arrive and the ideal number of configurations per method will both increase. Any mismatch in speed between the network and the RAP can be somewhat compensated for by the input and output buffers, but in the worst case may back up the network.

6 Prototype Hardware

A RAP test chip (Figure 7) has been fabricated and tested in MOSIS 3μm Scalable CMOS technology by MIT students Stuart Fiske, Josef Shaoul, and Petr Spacek in order to investigate some of the ideas described above, in particular the idea of having a reconfigurable data path. It consists of 12, 16-bit, two-bit serial, fixed point arithmetic units connected by statically reconfigurable sparse crossbar switches. The datapath is a three stage pipeline, each stage uses 4 AUs and is connected to the next stage by a switch. Each AU takes three operands and is capable of doing multiplication, addition, subtraction of two of its operands while passing the third unchanged, or of multiplying two of its operands and adding/subtracting the third. Two register files store input and output operands and perform parallel-serial and serial-parallel conversion.

Although the switch setup is different than that of the floating point RAP, this chip demonstrates that the switch can be efficiently implemented: about 12% of the total chip area is devoted to the switch and switch control, and this percentage will be much smaller in the case of the 64 bit floating point operations because the AUs and registers will be much bigger.

# Operands	Time for a set of operands to arrive	Best Number of Configurations
1	160ns	1
2	320ns	1
3	480ns	1
4	640ns	1
5	800ns	2
6	960ns	2
7	1.12μs	2
8	1.28μs	3
9	1.44μs	3
10	1.60μs	4
11	1.76μs	4
12	1.92μs	4
13	2.08μs	5
14	2.24μs	5
15	2.40μs	5
16	2.56μs	6

Table 2: Optimum number of configurations per method vs. number of input operands

7 Conclusion

The Reconfigurable Arithmetic Processor is a a special purpose processor specifically designed to fit into a message passing concurrent computer system. The RAP attempts to capitalize on the simplicity of serial arithmetic and the performance benefits of functional parallelism to create a powerful floating point arithmetic single chip processor. By changing the way in which its serial arithmetic units are interconnected, the RAP can provide additional flexibility by allowing complete arithmetic formulas to be calculated all at once without intermediate results going off chip or to local memory, substantially reducing the data transfer required.

The concepts used in the RAP provide the means to efficiently use message passing to achieve high performance numerical computing. A 1024 node message passing concurrent computer system with 128 RAPs would provide over 2GFlops of peak computing power.

Much work remains to be accomplished at the implementation level. The design of the serial floating point units in particular is critical and raises several issues related to the numerical aspects of floating point including handling *overflow*, *underflow*, rounding, and denormalised numbers. There is also the potential of pipelining two problems through each floating point unit in order to increase performance.

Areas for further research include investigating ways of implementing division in the same framework, as well as investigating how to take advantage of local memory in a RAP to further reduce the bandwidth constraints. In particular, allowing constants to be specified as part of a method would reduce I/O bandwidth in many cases. Giving the RAP more control over sequencing so that it is less dependent on off chip control may also lead to reduced bandwidth requirements and increased flexibility.

References

[1] Dally W.J., "A High Performance VLSI Quaternary Serial Multiplier", *Proc. ICCD-87*, pp. 649-653.

[2] Dally, W. J. et.al., "Architecture of a Message-Driven Processor," *Proceedings of the 14th ACM/IEEE Symposium on Computer Architecture*, June 1987, pp. 189-196.

[3] Dally W.J., et.al., "Concurrent Computer Architecture" *Proc. of Symp. on Parallel Computations and Their Impact on Mechanics*, 1987.

[4] Dally W.J., Seitz C.L., " The Torus Routing Chip", *J. Distributed Systems*, Vol. 1, No. 3, 1986, pp. 187-196.

[5] Dally W.J., Song P., "Design of a Self-Timed VLSI Multicomputer Communication Controller", *Proc. ICCD-87*, pp. 230-234.

[6] Denyer P., Renshaw W., *VLSI Signal Processing: A Bit-Serial Approach*, Addison-Wesley Publishing Company, 1985.

[7] Gosling J.B., Zurawski J.H.P., Edwards D.B.J., "A Chip Set for High-Speed Low Cost Floating Point Unit", *Proc. 5th Symposium on Computer Arithmetic*, IEEE Computer Society Press, 1981, pp. 50-55.

[8] Lyon R.F., "A Bit-Serial VLSI Architectural Methodology for Signal Processing", VLSI'81, ed. J.P. Gray, Academic Press, 1981, pp. 131-140.

[9] Lyon R.F., "MSSP: A Bit-Serial Multiprocessor for Signal Processing" *VLSI Signal Processing: A Bit-Serial Approach*, Denyer, Renshaw, Chapter 12, Addison-Wesley Publishing Company, 1985.

[10] Lyon R.F., "Two's Complement Pipeline Multipliers", *IEEE Trans. Comm.*, Vol COM-24, April 1976, pp. 418-425.

[11] McAllister W.H., Carlson J.R., "Floating-Point Chip Set Speeds Real-Time Computer Operation", *Hewlett-Packard Journal*, February 1984, pp. 17-23.

[12] Oppenheim A.V., Schafer R.W., *Digital Signal Processing*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.

[13] Owens R.M., "Compound Algorithms for Digit On-line Arithmetic", *5th Symp. Comput. Arith.*, Ann Arbor, MI, May 1981, pp. 64-71.

[14] Rabiner L.W., Gold B., *Theory and Applications of Digital Signal Processing*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.

[15] Rauch K., "Math Chips: how they work", *IEEE Spectrum*, July 1987, pp. 25-30.

[16] Trivedi K.S., Ercegovac M.D., "On-line Algorithms for Division and Multiplication", *IEEE Trans. Comput.*, vol. C-26, no. 7, pp.681-687, July 1977.

[17] Weitek Corporation, "WTL1064/1065 High Speed 64-bit IEEE Floating Point Multiplier/ALU" Preliminary Data 1984.